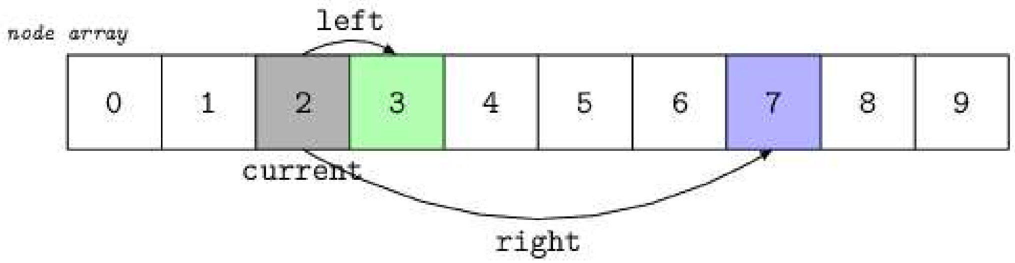
Kd-Trees

The objective of this assignment is to create a data structure that allows for fast ray intersection tests. This kind of structures are very useful for applications such as offline ray-tracing renderers.

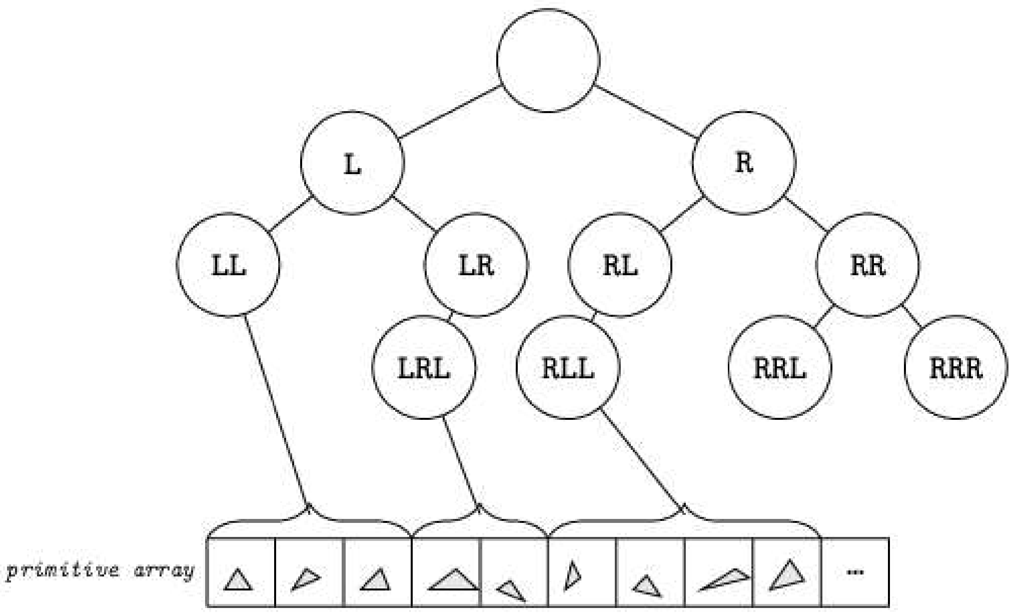
- The objective of this data structure is to save cost when traversing the Kd-Tree
- Building the Kd-Tree will be much more expensive than traversing it.
- It must always report the closest triangle to the ray, otherwise it will be considered incorrect.

Recommended node structure

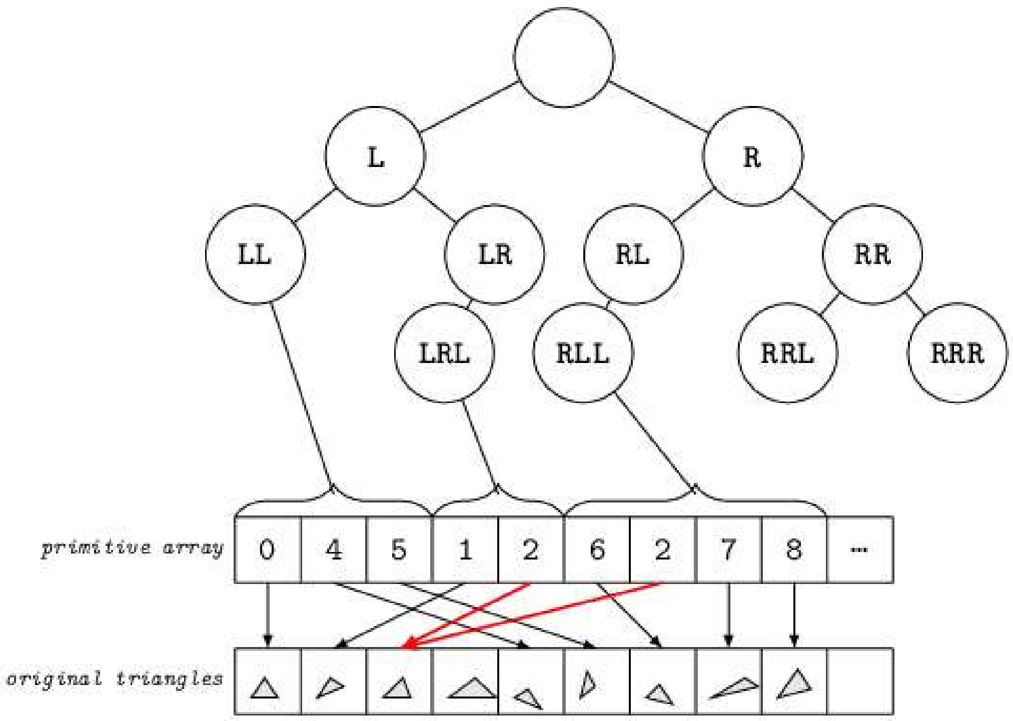
- Kd-Tree nodes should take **8 bytes exactly**. No more.
- **Internal** nodes should represent:
  - Split point: *Where did the separation occur?*
  - Split axis: *In which axis did the separation occur?*
  - Index to the right child: *How do I traverse to the right child node?* (e.g. 7, if this node is at index 2, left child is at index 3, and right child could be at index 7, so it needs to be stored)



  - This implies that you will need to build first all your left nodes before creating your right nodes
- **Leaf** nodes should represent:
  - The index to the first primitive index: *Where is the first* `primitive` *that this node contains?* (e.g. 62)
  - A count of primitives: *How many others are contiguously contained in this?* (e.g. 20, following previous example, this would contain triangles 62-82)
  - **The primitive list could be a list of triangles** (if triangles are not repeated)



- **The primitive list could be a list of triangle indices** (if triangles could appear in different nodes, and we want to avoid geometry duplication)



- It is recommended to also have a vector of AABBs for the nodes, in case that their bounding volume is tight. Store these in a separate vector, with the same index as the node.

Splitting

Several choices are provided to select a splitting point, regardless of the option, a heuristic formula will be used:

- **Option A**:
    - You are given the AABB of the current node (if not, compute it)
    - Try N positions uniformly (e.g. N=100) inside the AABB
        - For each one of those positions, use the heuristic formula to compute the cost
        - Keep the minimum cost as the splitting point
- **Option B**: For the splitting point follow the next strategy:
    - You are given all the triangles of the current node
    - Try all the triangle endpoints of the current axis
        - For each one of those positions, use the heuristic formula to compute the cost
        - Keep the minimum cost as the splitting point
- **Option C**: Free style. As long as it makes geometrical sense. Document it properly and **discuss it with the instructor prior implementation**.

Queries (extra credit: +20%)

Whenever we are querying intersections, sometimes we need to check both children nodes, sometimes it's not needed, see Physically Based Rendering (4.4.3). This varies on the chosen strategy straddle choice.

There are several tests, named `*_efficiency_*` which are meant to test the queries. These queries should be somewhat optimal, and considerably faster than the brute force approach.

How to submit

- You must include ⚠️ **generated images of the trees as the one above, use the `dot` tool** ⚠️ This allows for easy and fast detection of issues
- General submission guidelines apply: See guideline

FAQ and tips

Is an important question missing in this section?

Ask me and I will review it and probably add it here.

Maximum depth

The maximum depth configuration has been given as a way to stop early. You rarely should be making kdtrees that are vert deep!! **If the maximum depth is 50 and your `dragon` KdTree has a depth of 50, something is smelly**.

File: gam-assets.zip

Comments

Loading comments...